I Erläuterungen

Voraussetzungen gemäß KCBG und Abiturerlassen BG jeweils in der für den Abiturjahrgang geltenden Fassung

Standardbezug
Die nachfolgend ausgewiesenen Kompetenzbereiche sind für die Bearbeitung der jeweiligen Aufgabe besonders bedeutsam. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Kompetenzen für die Bearbeitung der Aufgabe nachrangig bedeutsam sein, zumal die Kompetenzen in engem Bezug zueinander stehen. Die Operationalisierung des Bezugs zu den Kompetenzbereichen des Standardbezugs erfolgt in Abschnitt II.

Aufgabe	Kompetenzen				
	K1	K2	K3	K4	K5
1.1	X	X			
1.2		X			X
1.3.1				X	
1.3.2				X	
1.3.3	X		X	X	
1.3.4			X	X	
1.4		X	X		
2.1.1	X	X			
2.1.2		X		X	
2.1.3			X	X	
2.1.4			X	X	
2.2		X	X		
2.3.1				X	
2.3.2	X			X	
2.3.3		X	X		
2.3.4		X	X		

Inhaltlicher Bezug
Die nachfolgend ausgewiesenen Themenfelder sind die wesentliche inhaltliche Grundlage für die vorliegenden Aufgaben. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Themenfelder für die Bearbeitung nachrangig bedeutsam sein.
Q1: Objektorientierte Softwareentwicklung
Q2: Datenbanksysteme
Q3: Datenkommunikation
verbindliche Themenfelder: Objektorientierte Modellierung (Q1.1), Implementierung von Klassen und Assoziationen (Q1.2), Konzeptionelle und logische Modellierung einer Datenbank (Q2.1), Datenabfrage und Datenmanipulation mit SQL (Q2.2), Kommunikation in Rechnernetzen (Q3.2)

II Lösungshinweise

In den nachfolgenden Lösungshinweisen sind alle wesentlichen Gesichtspunkte, die bei der Bearbeitung der einzelnen Aufgaben zu berücksichtigen sind, konkret genannt und diejenigen Lösungswege aufgezeigt, welche die Prüflinge erfahrungsgemäß einschlagen werden. Selbstverständlich sind jedoch Lösungswege, die von den vorgegebenen abweichen, aber als gleichwertig betrachtet werden können, ebenso zu akzeptieren.

Aufg.	**erwartete Leistungen**	**BE**		
		I	**II**	**III**

1.1	überführen
	`Linie(lid, Takt)`
	`Segment(sid, vmax, laenge, von#, nach#)`
	`Verbindet(sid#, lid#, positionsnr)`
	`Bahnhof(bid, name)`
	`Haelt_in(bid#, zid#, abfahrt, ankunft)`
	
	`Zug(zid, lid#, startbid#, abfahrt, zielbid#, ankunft)`
	
	erläutern
	Alle Entitätstypen werden mit ihren Attributen in Relationen überführt.
	Die 1:n-Beziehung `von` zwischen `Bahnhof` und `Segment` wird abgebildet, indem der PK der 1-Relation (`Bahnhof`) als FK bei der n-Relation (`Segment`) eingetragen wird. Analog wird auch bei den 1:n-Beziehungen `nach` zwischen `Bahnhof` und `Segment` sowie `faehrt` zwischen `Linie` und `Zug` verfahren.
	Die n:m-Beziehungen `verbindet` und `haelt in` werden mit Zwischentabellen realisiert, die neben den PKs der verbundenen Relationen auch die Attribute der Beziehungen aufnehmen.
	Die attributbehafteten 1:n-Beziehungen `startet in` und `faehrt nach` zwischen `Bahnhof` und `Zug` werden umgesetzt, indem die PKs der Relation `Bahnhof` sowie die Attribute `abfahrt` und `ankunft` in die Relation `Zug` aufgenommen werden. Beide PKs sind dann als Fremdschlüssel zu kennzeichnen.
		3	2
			
			3	

1.2	analysieren, diskutieren Die Modelle unterscheiden sich grundsätzlich in der Art der Speicherung von Ankunft- und Abfahrtzeiten. ERM A speichert alle Ankunft- und Abfahrtzeiten für jeden Zug und jeden Bahnhof explizit. Das Modell stellt daher hohe Anforderungen an den Speicherbedarf der Datenbank. Der Zugriff auf Ankunft- bzw. Abfahrtzeiten erfolgt direkt über Attribute. ERM B nutzt das Konzept des Taktbetriebs der Linien aus. Hier werden für jede Linie nur die Abfahrtminuten gespeichert, da diese aufgrund des Taktbetriebs für alle Züge identisch sind. Der einzelne Zug kennt die Stunde seiner Abfahrt. Aus diesen beiden Informationen kann die Abfahrtzeit eines Zuges ermittelt werden. Des Weiteren werden über die `verbindet`-Beziehung die Fahrzeiten der Streckenabschnitte gespeichert. Die Haltezeiten sind bahnhofsbezogen und werden in `Bahnhof` gespeichert. Mit diesen Zeitinformationen können alle Abfahrt- und Ankunftszeiten berechnet werden. Der Speicherbedarf des ERM B ist geringer als bei ERM A. Allerdings steigt der Rechenaufwand beim Zugriff auf die Zeitinformationen, weil viele Unterabfragen erforderlich sind. ERM B erweist sich bei Fahrplanänderungen als vorteilhaft, weil nur wenige Attribute (Abfahrtminuten, ggf. Abfahrtstunden) zu aktualisieren sind. In ERM A müssen dagegen sehr viele Datenbankeinträge verändert werden, so dass das Risiko von Anomalien größer ist. analysieren diskutieren		3	3
1.3.1	angeben `SELECT MAX(laenge) AS 'längste Teilstrecke', SUM(laenge) AS Gesamtlänge FROM Segment;`	2		
1.3.2	angeben `DELETE FROM Zug WHERE zid = 'ICE 944';`	2		

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.3.3	angeben (siehe unten)	4		
	implementieren (siehe unten)		5	2
1.3.4	implementieren (siehe unten)		2	3
1.4	entwickeln		2	2
	zeichnen	2		
	Summe 40	**13**	**17**	**10**

1.3.3 angeben

```
# Neuen Bahnhof einfügen
INSERT INTO Bahnhof VALUES('BA', 'Basel', 5);

# Segmente für neues Teilstück in beide Richtungen einfügen
INSERT INTO Segment VALUES
(null, 70, 250, 'FR', 'BA'),
(null, 70, 250, 'BA', 'FR');

# Segment an erster Position von Linie 26B einfügen
INSERT INTO Verbindung VALUES
((SELECT sid FROM Segment WHERE von = 'BA' AND nach = 'FR'),
'26B', 0, 35);
```

implementieren

```
# Abfahrtminuten Linie 26B aus Basel setzen
UPDATE Linie SET abfahrtminuten = 7
WHERE lid = '26B';

# Position der Verbindungselemente von 26B um 1 erhöhen
UPDATE Verbindung SET positionsnr = positionsnr + 1
WHERE lid = '26B';

# Segment an Linie 26A anfügen
INSERT INTO Verbindung VALUES
((SELECT sid FROM Segment WHERE von = 'FR' AND nach = 'BA'),
'26A', 5, 35);
```

1.3.4 implementieren

```
SELECT lid AS Linie, zid AS Zug,
MAKETIME(z.abfahrtstunde, l.abfahrtminuten, 0) AS Abfahrt,
(SELECT name FROM Bahnhof WHERE bid = s.nach) AS Nach
FROM Bahnhof JOIN Segment s ON bid = s.von
JOIN Verbindung v USING(sid)
JOIN Linie l USING(lid)
JOIN Zug z USING(lid)
WHERE name = 'München'
ORDER BY Abfahrt;
```

1.4 entwickeln, zeichnen

Zug
[0,n]
[0,n]
datum
zieht
hat
datum
[2,2]
[1,n]
inbetriebnahme
inbetriebnahme
Triebwagen
Waggon
wid
twid
klasse
[0,n]
kid
[1,1]
Kategorie
gehört zu
bezeichnung

entwickeln
zeichnen

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.1.1	nennen, erläutern Einfache Assoziationen: Zwischen den Klassen `Route` und `Bahnhof` besteht eine einfache unidirektionale Assoziation. Die Route kennt den Bahnhof (Rollenname `start`), mit dem sie beginnt, der Bahnhof kennt die Route jedoch nicht, weil diese Navigationsrichtung explizit ausgeschlossen ist. Die Multiplizität bei der Klasse `Bahnhof` ist eins. Es handelt sich um eine Muss-Beziehung. Die Assoziation wird durch eine Referenzvariable des jeweiligen Zieltyps implementiert. Die Klasse `Route` erhält eine Referenzvariable vom Typ `Bahnhof`. Zwischen den Klassen `Bahnhof` und `Segment` besteht eine unidirektionale Eins-zu-viele-Beziehung. Von einem Bahnhof führen mehrere Strecken zu benachbarten Bahnhöfen. Assoziationen mit der Multiplizität viele werden mit Containern von Referenzvariablen des Typs der Zielkasse umgesetzt. Der Bahnhof besitzt eine Sammlung von Referenzvariablen des Typs `Segment`. Aggregation: Ein Streckensegment ist Bestandteil einer Route, die sich aus vielen Segmenten zusammensetzt. Diese Beziehung zwischen einem Aggregat (Route) und seinen Teilen (Segmente) wird durch eine Aggregation zwischen den Klassen `Route` und `Segment` abgebildet. Komposition: Das Streckennetz (Aggregat) besteht aus Bahnhöfen und Streckensegmenten, die die Teile des Netzes bilden. Sie sind untrennbar mit dem Streckennetz verbunden. Dieser Sachverhalt wird durch die Komposition zwischen den Klassen `Streckennetz` und `Bahnhof` sowie `Streckennetz` und `Segment` abgebildet. Beide Kompositionen beschreiben 1-zu-viele-Beziehungen. Aggregationen werden in derselben Weise implementiert wie einfache Assoziationen. Bei Kompositionen werden die Teile in der Regel durch das Aggregat erzeugt.			
	nennen	3		
	erläutern		3	

2.1.2	überführen, implementieren
	`public class Bahnhof {`
	`  private String name;`
	`  private String bid;`
	`  private List<Segment> segmente;`
	
	`  public Bahnhof(String bid, String name) {`
	`    this.bid = id;`
	`    this.name = name;`
	`    this.segmente = new List<>();`
	`  }`
	`  public boolean hinzufuegen(Segment segment) {`
	`    if (!segmente.contains(segment)) {`
	`      segmente.add(segment);`
	`      return true;`
	`    }`
	`    return false;`
	`  }`
	`}`			

	überführen	2		
	implementieren		2	

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.1.3	implementieren	1	4	

```java
public class Streckennetz {
  private List<Bahnhof> bahnhoefe = new List<>();
  private List&lt;Segment&gt; segmente = new List<>();

  public Route findeKuerzesteRoute(String start,
                                   String ziel) {
    Bahnhof von = null, nach = null;
    for (Bahnhof b : bahnhoefe) {
      if (b.getName().equals(start)) {
        von = b;
      }
      else if (b.getName().equals(ziel)) {
        nach = b;
      }
    }
    if (von != null && nach != null && von != nach) {
      return suchen(new Route(von), nach, null);
    }
    return null;
  }
}
```

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.1.4	implementieren			5

```java
private Route suchen(Route aktuell, Bahnhof ziel,
                                      Route kuerzeste) {
  List&lt;Segment&gt; segmente =
                   aktuell.ermittleZiel().getSegmente();
  for (Segment s : segmente) {
    if (!aktuell.enthaelt(s.getZiel())) {
      aktuell.anfuegen(s);
      if (s.getZiel() == ziel) {
        if (kuerzeste == null || aktuell.berechneLaenge() <
                               kuerzeste.berechneLaenge()) {
          kuerzeste = aktuell.kopieren();
        }
      } else {
          kuerzeste = suchen(aktuell, ziel, kuerzeste);
      }
      aktuell.entfernenLetztes();
    }
  }
  return kuerzeste;
}
```

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.2	modellieren, zeichnen			

Aufg.	erwartete Leistungen	I	II	III
	modellieren		4	2
	zeichnen	3		
2.3.1	überführen, implementieren			

```
public class FahrplanServer {
  private ServerSocket serverSocket;
  private Streckennetz netz;

  public FahrplanServer(int port, Streckennetz sn) {
    serverSocket = new ServerSocket(port);
    netz = sn;
  }

  public void starteServer() {
    while (true) {
      Socket s = serverSocket.accept();
      Verbindungsanfrage anfrage =
                     new Fahrplananfrage(s, netz, this);
      anfrage.start();
    }
  }
}
```

Aufg.	erwartete Leistungen	I	II	III
	überführen	2		
	implementieren		2	

Nicht für den Prüfling bestimmt

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.3.2	implementieren (siehe Code unten)	2	4	5

```
public class FahrplanClient {
  private Socket socket;

  public boolean verbinden(String server, int port) {
    socket = new Socket(server, port);
    return socket.connect();
  }

  public boolean anmelden(String email, String passwort,
                          boolean istRegistriert) {
    String aktion;
    if(istRegistriert)
      aktion = "login;";
    else
      aktion = "register;"
    socket.write(aktion + email + ";" + passwort + "\n");
    String antwort = socket.readLine();
    if (!antwort.startsWith("+OK")) {
      socket.close();
      return false;
    }
    return true;
  }

  public boolean anmeldenGast(){
    return anmelden("gast@fahrplan", "inkognito", true);
  }

  public boolean abmelden(){
    socket.write("logout\n");
    String antwort = socket.readLine();
    if (antwort.startsWith("+OK")) {
      socket.close();
      return true;
    }
    return false;
  }
  public String sucheBahnverbindung(String start,
              String ziel, DateTime abfahrt){
    socket.write("Suche Verbindung:Start:" + start +
                 ",Ziel:" + ziel + ",Abfahrt:" +
                 abfahrt.toString() + "\n");
    String verbindung = socket.readLine();
    return verbindung;
  }
}
```

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.3.3	entwickeln, zeichnen : Verbindung : Zug id = ICE 640 : Halt ankunft = null abfahrt = 10.06.2022 12:20 : Bahnhof name = Frankfurt : Zug id = ICE 730 : Halt ankunft = 10.06.2022 12:48 abfahrt = 10.06.2022 13:05 : Bahnhof name = Mannheim : Halt ankunft = 10.06.2022 14:05 abfahrt = null : Bahnhof name = Stuttgart			
	entwickeln		3	2
	zeichnen	1		

Aufg.	**erwartete Leistungen**	**BE** I	**BE** II	**BE** III
2.3.4	modellieren, zeichnen : Fahrplananfrage : Socket : FahrplanServer : Streckennetz run() write("OK+ connected\n") readLine() {antwort} login(mail, passwort) write("OK+ login\n") readLine() {anfrage} findeVerbindung(von, nach, abfahrt) findeKuerzesteRoute(von, nach) {route} <<create>> Verbindung(route.start) : Verbindung toString() {vString} write("OK+ "+vString+"\n") readLine() write("OK + logout\n")			
	modellieren		3	4
	zeichnen	3		
	Summe 60	**17**	**25**	**18**

III Bewertung und Beurteilung

Die Bewertung und Beurteilung erfolgt unter Beachtung der nachfolgenden Vorgaben nach § 33 der Oberstufen- und Abiturverordnung (OAVO) in der jeweils geltenden Fassung. Bei der Bewertung und Beurteilung der sprachlichen Richtigkeit in der deutschen Sprache sind die Bestimmungen des § 9 Abs. 12 Satz 3 OAVO in Verbindung mit Anlage 9b anzuwenden.

Bei der Bewertung und Beurteilung der Übersetzungsleistung in den Fächern Latein und Altgriechisch sind die Bestimmungen des § 9 Abs. 14 OAVO in Verbindung mit Anlage 9c anzuwenden.

Der Fehlerindex ist nach Anlage 9b zu § 9 Abs. 12 OAVO zu berechnen. Für die Ermittlung der Punkte nach Anlage 9a zu § 9 Abs. 12 OAVO sowie Anlage 9c zu § 9 Abs. 14 OAVO wird jeweils der ganzzahlige nicht gerundete Prozentsatz bzw. Fehlerindex zugrunde gelegt.

Für die Bewertung in den modernen Fremdsprachen ist der „Erlass zur Bewertung und Beurteilung von schriftlichen Arbeiten in allen Grund- und Leistungskursen der neu beginnenden und fortgeführten modernen Fremdsprachen in der gymnasialen Oberstufe, dem beruflichen Gymnasium, dem Abendgymnasium und dem Hessenkolleg“ vom 7. August 2020 (ABl. S. 519) zugrunde zu legen. Demnach erfolgt die Bewertung und Beurteilung mit der Maßgabe, dass lediglich bei der Ermittlung des Prüfungsergebnisses (Note) aus Prüfungsteil 1 und 2 gerundet wird.

Darüber hinaus sind die Vorgaben der Erlasse „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen (Abiturerlass)“ und „Durchführungsbestimmungen zum Landesabitur“ in der für den Abiturjahrgang geltenden Fassung zu beachten.

Als Kriterien für die Bewertung und Beurteilung dienen unter Beachtung der Zielsetzung der gymnasialen Oberstufe nach § 1 Abs. 2 OAVO neben dem Inhaltlichen auch die in den Kerncurricula genannten überfachlichen Kompetenzen, insbesondere die Sprachkompetenz und Wissenschaftspropädeutik; dies zeigt sich u.a. in qualitativen Merkmalen wie Strukturierung, Differenziertheit, (fach-)sprachlicher Gestaltung und Schlüssigkeit der Argumentation.

Im Fach Praktische Informatik besteht die Prüfungsleistung aus der Bearbeitung eines Vorschlags, wofür insgesamt maximal 100 BE vergeben werden können. Ein Prüfungsergebnis von **5 Punkten (ausreichend)** setzt voraus, dass mindestens 45% der zu vergebenden BE erreicht werden. Ein Prüfungsergebnis von **11 Punkten (gut)** setzt voraus, dass mindestens 75% der zu vergebenden BE erreicht werden.

Gewichtung der Aufgaben und Zuordnung der Bewertungseinheiten zu den Anforderungsbereichen

Aufgabe	**Bewertungseinheiten in den Anforderungsbereichen**			**Summe**
	AFB I	**AFB II**	**AFB III**	
1	13	17	10	**40**
2	17	25	18	**60**
Summe	**30**	**42**	**28**	**100**

Die auf die Anforderungsbereiche verteilten Bewertungseinheiten innerhalb der Aufgaben sind als Richtwerte zu verstehen.